**Tugas Kecil 1 IF2211 Strategi Algoritma**

**Penyelesaian Cyberpunk Breach Protocol menggunakan metode Brute Force**

A black background with white lines

Description automatically generated

Disusun oleh :

13522153 – Muhammad Fauzan Azhim

INSTITUT TEKNOLOGI BANDUNG 2023

Daftar Isi

[Daftar Isi 1](#_TOC_250016)

[BAB I 2](#_TOC_250015)

* 1. [Algoritma Brute Force 2](#_TOC_250014)
  2. Cyberpunk Breach Protokol [2](#_TOC_250013)

[BAB II 3](#_TOC_250012)

* 1. [Library 3](#_TOC_250011)
  2. [Algoritma 3](#_TOC_250010)

[BAB III 4](#_TOC_250006)

* 1. [Repository Program 4](#_TOC_250005)
  2. [Source Code Program 4](#_TOC_250004)

[BAB IV 15](#_TOC_250003)

[REFERENSI 16](#_TOC_250000)

## BAB I

### DESKRIPSI MASALAH DAN ALGORITMA

##### Algoritma Brute Force

Algoritma brute force adalah jenis algoritma yang mencoba semua kemungkinan untuk menyelesaikan masalah. Dengan kata lain, ia akan melakukan percobaan satu persatu hingga menemukan solusi yang tepat. Contoh penggunaan algoritma Brute Force misalnya untuk memecahkan 4 digit kode PIN dari angka 0 hingga 9.

Meskipun algoritma bruteforce dapat menjamin untuk menemukan solusi jika ada, pendekatan ini seringkali tidak efisien dan memakan waktu lama, terutama untuk masalah dengan ruang solusi yang sangat besar. Oleh karena itu, algoritma bruteforce biasanya hanya digunakan jika ukuran masalah cukup kecil atau jika semua kemungkinan lain telah dicoba dan tidak berhasil.

##### Cyberpunk Breach Protocol

##### Cyberpunk breach protocol adalah minigame dalam cyberpunk yang dilakukan saat player sedang melakukan hacking. Game ini dimulai dengan memunculkan Matrix, Buffer, dan sequence. Target dari minigame ini adalah mengisi buffer menggunakan angka-angka matrix yang dipilih oleh player.

Pemilihan dimulai dari baris paling atas, lalu dipilih melalui kolom yang sama dengan pilihan pertama. Lalu dipilih melalui baris yang sama dengan pilihan kedua, dan seterusnya berulang-ulang. Untuk setiap sequence yang terdapat pada buffer, player akan diberikan hadiahnya sesuai dengan sequence yang diselesaikan. Sequence yang diselesaikan bisa lebih dari satu.

## BAB II

### IMPLEMENTASI ALGORITMA DALAM BAHASA C++

Dalam pembuatan program ini, penulis menggunakan bahasa pemrograman C++. Dan menggunakan Qt Creator dalam pembuatannya.

2.1. Library

Adapun library yang diganakan adalah berikut:

#include <iostream>

#include <fstream>

#include <vector>

#include <string>

#include <iostream>

#include <fstream>

#include <utility>

#include <random>

#include <chrono>

#include <cstring>

#include <cerrno>

#include <algorithm>

2.2 Algoritma

Dalam pembuatan algoritma BruteForce ini, saya mengimplementasikan rekursif untuk mencari semua kemungkinan yang ada dan mengoptimisasi denan tidak mencari kemungkinan yang tidak akan pernah terjadi dengan beberapa handle. Seperti melakukan limit pada kedalaman rekursif dengan ukuran buffer.

Program dimulai dengan kedalaman 0, dicari semua kemungkinan dengan rekursifitas. Kemudian disaring kemana saja gerakan selanjutnya dapat dilakukan. Setelah itu kita loop gerakan-gerakan selanjutnya itu dan mencari gerakan dengan score terbesar dan sequence terkecil. Setelah selesai dilakukan , lakukan rekursifitas dan bergerak menggunakan gerakan yang paling maksimal efisiensinya , yang telah kita temukan sebelumnya.

## BAB III

### SOURCE CODE PROGRAM

##### Repository Program

Repository program dapat diakses melalui tautan *GitHub* berikut :

[fauzanazz/Tucil1\_13522153 (github.com)](https://github.com/fauzanazz/Tucil1_13522153)

3.2 Sourcecode Program utama

class Matrix {

private:

    int rows;

    int cols;

    vector<vector<int>> data;

public:

    [[nodiscard]] int getRows() const {

        return rows;

    }

    [[nodiscard]] int getCols() const {

        return cols;

    }

    [[nodiscard]] int getMatrixData(int *r*, int *c*) const {

        return data[*r*][*c*];

    }

    [[nodiscard]] vector<vector<int>>& getData() {

        return data;

    }

*/// @brief Get Data from Matrix*

*/// @param r row index*

*/// @param c column index*

*/// @return Data that the matrix has*

    [[nodiscard]] int GetMatrixData(int *r*, int *c*) const {

        return data[*r*][*c*];

    }

*/// @brief Set Data to Matrix*

*/// @param r row index*

*/// @param c col index*

*/// @param value the value that will be set to the matrix*

    void SetMatrixData(int *r*, int *c*, int *value*){

        data[*r*][*c*] = *value*;

    }

*/// @brief Set Data to Matrix, the inputted matrix will be set to the matrix size if the setter is larger*

*/// @param setData Matrix that will be set to the matrix*

    void SetMatrixData(const vector<vector<int>>& *setData*){

        for (int i = 0; i < rows; i++) {

            for (int j = 0; j < cols; j++) {

                SetMatrixData(i, j, *setData*[i][j]);

            }

        }

    }

*/// @brief Matrix Constructor*

*/// @param r sum row*

*/// @param c sum column*

*/// @param GenerateRandom boolean to generate random matrix with a set of value*

*/// @param setData matrix that will be set to the matrix (if GenerateRandom is false)*

    Matrix(int *r*, int *c*, bool *GenerateRandom* = true, const vector<vector<int>>& *setData* = {}) {

        rows = *r*;

        cols = *c*;

        data.resize(rows, vector<int>(cols));

        const vector<int> allowedDigits = {0x1C, 0x55, 0x7A, 0xBD, 0xE9, 0xFF};

        if (*GenerateRandom*) {

            const auto seed = chrono::system\_clock::now().time\_since\_epoch().count();

            default\_random\_engine generator(seed);

            uniform\_int\_distribution<vector<int>::size\_type> distribution(0, allowedDigits.size() - 1);

            for (int i = 0; i < rows; i++) {

                for (int j = 0; j < cols; j++) {

                    data[i][j] = allowedDigits[distribution(generator)];

                }

            }

        } else if (!*setData*.empty()) {

            for (int i = 0; i < rows; i++) {

                for (int j = 0; j < cols; j++) {

                    data[i][j] = *setData*[i][j];

                }

            }

        }

    }

};

*/// @brief Sequence class*

class Sequence {

    vector<int> data;

public:

    Sequence() = default;

*/// @brief Sequence Constructor*

*/// @param len length of the sequence*

*/// @param GenerateRandom boolean to generate random sequence with a set of value*

*/// @param setData sequence that will be set to the sequence (if GenerateRandom is false)*

    explicit Sequence(int *len*, const bool *GenerateRandom* = true, const vector<int>& *setData* = {}) {

        data.resize(*len*);

        if (*GenerateRandom*) {

            const auto seed = chrono::system\_clock::now().time\_since\_epoch().count();

            default\_random\_engine generator(seed);

            uniform\_int\_distribution<vector<int>::size\_type> distribution(0, 255);

            for (int i = 0; i < *len*; i++) {

                data[i] = distribution(generator);

            }

        } else if (!*setData*.empty()) {

            for (int i = 0; i < *len*; i++) {

                data[i] = *setData*[i];

            }

        }

    }

*/// @brief Set Data to Sequence*

*/// @param index index of the sequence*

*/// @param value the value that will be set to the sequence*

    void SetData(int *index*, int *value*){

        data[*index*] = *value*;

    }

*/// @brief Set Data to Sequence, the inputted sequence will be set to the sequence size if the setter is larger*

*/// @param setData sequence that will be set to the sequence*

    void SetData(const vector<int>& *setData*){

        for (int i = 0; i < data.size(); i++) {

            SetData(i, *setData*[i]);

        }

    }

    [[nodiscard]] int getData(int *index*) const{

        return data[*index*];

    }

    [[nodiscard]] size\_t getLength() const {

        return data.size();

    }

};

class Data {

private:

    int buffer\_size{};

    int matrix\_width, matrix\_height;

    Matrix matrix;

    int number\_of\_sequences;

    vector<Sequence> sequences;

    int buffer;

public:

    void printSequences() const {

        for (int i = 0; i < number\_of\_sequences; i++) {

            cout << "Sequence " << i << ": ";

            for (int j = 0; j < sequences[i].getLength(); j++) {

                cout << hex << sequences[i].getData(j) << " ";

            }

            cout << endl;

        }

    }

    void printMatrix () const {

        for (int i = 0; i < matrix\_height; i++) {

            for (int j = 0; j < matrix\_width; j++) {

                cout << hex << matrix.GetMatrixData(i, j) << " ";

            }

            cout << endl;

        }

    }

*/// @brief Read data from file*

*/// @param filename*

    void readFromFile(const string& *filename*) {

        ifstream file(*filename*);

        if (!file) {

            cout << "Unable to open file: " << *filename* << endl;

            cout << "Error code: " << strerror(errno) << endl;

            return;

        }

        file >> buffer\_size;

        file >> matrix\_width >> matrix\_height;

        matrix = Matrix(matrix\_width, matrix\_height);

        for (int i = 0; i < matrix\_height; i++) {

            for (int j = 0; j < matrix\_width; j++) {

                string hexValue;

                file >> hexValue;

                matrix.SetMatrixData(i, j, stoi(hexValue, nullptr, 16));

            }

        }

        file >> number\_of\_sequences;

        sequences.resize(number\_of\_sequences, Sequence());

        for (int i = 0; i < number\_of\_sequences; i++) {

            string hexValue;

            file >> hexValue;

            sequences[i].SetData(0, stoi(hexValue, nullptr, 16));

        }

    }

    Data() : buffer(10), matrix(10, 10) {

        matrix\_width = 10;

        matrix\_height = 10;

        matrix = Matrix(matrix\_width, matrix\_height);

        number\_of\_sequences = 5;

        for (int i = 0; i < number\_of\_sequences; i++) {

            sequences.emplace\_back(5);

        }

    }

    ~Data() {

        while (!sequences.empty()) {

            sequences.pop\_back();

        }

    }

    pair<vector<int>, vector<vector<int>>> solve() {

        return findMoves(matrix.getData(), {0, 0}, {{}, {}, {}}, vector<int>(1, 0), 1, buffer\_size);

    }

*/// @brief update the score of the game in a row*

*/// @param row the row that is being check*

*/// @param built\_sequences the built sequences*

*/// @param sequences sequences that is being built*

*/// @return the score and the updated sequences*

    static pair < vector<int>, vector<vector<int>> > updateScore (vector<int> *row*, vector<vector<int>> *built\_sequences*, vector<vector<int>> *sequences*) {

        vector<int> score(*row*.size(), 0);

        vector<vector<int>> updated\_sequences(*row*.size(), vector<int>());

*// Copy the built sequences to the updated sequences*

        for (int i = 0; i < *row*.size(); i++) {

            updated\_sequences[i] = *built\_sequences*[i];

        }

*// iterate through the row and column*

        for (int i = 0; i < *row*.size(); i++) {

            for (int j = 0; j < *built\_sequences*.size(); j++) {

                vector<int> target\_sequence = *sequences*[j];

*// Check if the current sequence is still being built*

                if (target\_sequence.size() > *built\_sequences*[j].size() && *row*[i] == target\_sequence[*built\_sequences*[j].size()]) {

                    updated\_sequences[i].push\_back(*row*[i]);

                    if (updated\_sequences[i].size() < target\_sequence.size()) {

                        score[i] += j + 1;

                    } else {

                        score[i] += (j + 1) \* 9;

                    }

                } else {

*// If the current sequence is not being built, clear the sequence*

                    if (updated\_sequences[i].size() < target\_sequence.size()) {

                        updated\_sequences[i].clear();

                    }

                }

            }

        }

        return make\_pair(score, updated\_sequences);

    }

*/// @brief find the moves that is possible to do in recursion*

*/// @param game\_matrix the matrix*

*/// @param current\_position the current position*

*/// @param sequence\_built sequence that is being buit*

*/// @param max\_game\_score max game score*

*/// @param depth the depth of recursion*

*/// @param buffer\_size the buffer size*

*/// @return the max game score and the new positions*

    pair<vector<int>, vector<vector<int>>> findMoves(vector<vector<int>> *game\_matrix*, pair<int, int> *current\_position*, vector<vector<int>> *sequence\_built*, vector<int> *max\_game\_score*, int *depth*, int *buffer\_size*) {

        vector<pair<int, int>> new\_positions\_map;

        int current\_score = *max\_game\_score*[0];

        pair<int, int> next\_move = {0, 0};

*// Where to move, row or column*

        bool movementDecider = *depth* % 2 == 1;

        vector<int> game\_row;

*// If the movement decider is true, then the game row is the current row*

*// If the movement decider is false, then the game row is the current column*

        if (movementDecider) {

            game\_row = *game\_matrix*[*current\_position*.second];

        } else {

            for (auto & i : *game\_matrix*) {

                game\_row.push\_back(i[*current\_position*.first]);

            }

        }

*// Change vector<sequence> to vector<vector<int>>*

        vector<vector<int>> sequences;

        for (auto & i : sequences) {

            vector<int> temp;

            for (auto & j : i) {

                temp.push\_back(j);

            }

            sequences.push\_back(temp);

        }

*// Find the score and the sequences*

        pair<vector<int>, vector<vector<int>>> score\_and\_sequences = updateScore(game\_row, std::move(*sequence\_built*), sequences);

*// Get the maximum rewards*

        int maxRewards = \*max\_element(score\_and\_sequences.first.begin(), score\_and\_sequences.first.end());

*// Get the indexes of the maximum rewards*

        vector<int> yCords;

        vector<int> xCords;

        vector<int> maxScoreIndex = get\_indexes(score\_and\_sequences.first, maxRewards);

*// If the movement decider is true, then the yCords is the current row*

*// else the xCords is the current column*

        if (movementDecider) {

            yCords = vector<int>(maxScoreIndex.size(), *current\_position*.second);

            xCords = maxScoreIndex;

        } else {

            yCords = maxScoreIndex;

            xCords = vector<int>(maxScoreIndex.size(), *current\_position*.first);

        }

*// Change the xCords and yCords to a pair of x and y*

        vector<pair<int, int>> new\_positions;

        for (int i = 0; i < xCords.size(); i++) {

            new\_positions.emplace\_back(xCords[i], yCords[i]);

        }

*// Remove the current position from the new positions*

        auto it = find(new\_positions.begin(), new\_positions.end(), *current\_position*);

        if (it != new\_positions.end()) {

            new\_positions.erase(it);

        }

*// If the depth is greater than the buffer size, return the max game score and the new positions*

        if (*depth* >= *buffer\_size*) {

            return make\_pair(vector<int>(1, *max\_game\_score*[0] + maxRewards), new\_positions.empty() ? vector<vector<int>>() : vector<vector<int>>(1, {new\_positions[0].first, new\_positions[0].second}));

        }

*// Iterate through the new positions*

        for (auto& pos : new\_positions) {

            vector<vector<int>> new\_game\_matrix = *game\_matrix*;

            new\_game\_matrix[pos.second][pos.first] = 0;

            const vector<vector<int>>& new\_sequence = *sequence\_built*;

            auto [new\_maxRewards, temp\_positions\_map] = findMoves(new\_game\_matrix, pos, new\_sequence, vector<int>(1, maxRewards), *depth* + 1, *buffer\_size*);

*// If the new max rewards + the current score is greater than the max game score, then the max game score is the new max rewards + the current score*

            if (new\_maxRewards[0] + current\_score > *max\_game\_score*[0]) {

*max\_game\_score*[0] = new\_maxRewards[0] + current\_score;

                next\_move = pos;

                new\_positions\_map = convert\_to\_pairs(temp\_positions\_map);

            }

        }

*// If the next move is not empty, then the new positions map is the next move*

        new\_positions\_map.push\_back(next\_move);

        return make\_pair(vector<int>(1, *max\_game\_score*[0]), convert\_to\_vectors(new\_positions\_map));

    };

*/// @brief convert the sequence built to a vector of pair of int*

*/// @param pairs*

*/// @return vector of vector of int*

    static vector<vector<int>> convert\_to\_vectors(const vector<pair<int, int>>& *pairs*) {

        vector<vector<int>> vecs;

        for (auto &p : *pairs*) {

            vecs.push\_back({p.first, p.second});

        }

        return vecs;

    }

*/// @brief get indexes of the element in the vector*

*/// @param vec the vector*

*/// @param element element to find*

*/// @return indexes of the element in the vector*

    static vector<int> get\_indexes(vector<int> *vec*, int *element*) {

        vector<int> indexes;

        for (int i = 0; i < *vec*.size(); i++) {

            if (*vec*[i] == *element*) {

                indexes.push\_back(i);

            }

        }

        return indexes;

    }

*/// @brief Convert the sequence built to a vector of pair of int*

*/// @param vec the vector of sequence built*

*/// @return vector of pair of int*

    static vector<pair<int, int>> convert\_to\_pairs(vector<vector<int>> *vec*) {

        vector<pair<int, int>> pairs;

        for (auto &v : *vec*) {

            pairs.emplace\_back(v[0], v[1]);

        }

        return pairs;

    }

*/// @brief print all data that is stored in the class*

    void printAll() const {

        cout<<"Buffer Size: "<<buffer\_size<<endl;

        cout<<"Matrix Width: "<<matrix\_width<<endl;

        cout<<"Matrix Height: "<<matrix\_height<<endl;

        cout<<"Number of Sequence: "<<number\_of\_sequences<<endl;

        printMatrix();

        printSequences();

    }

};

## BAB IV

LAMPIRAN

|  |  |  |
| --- | --- | --- |
| Poin | Ya | Tidak |
| 1. Program berhasil dikompilasi tanpa kesalahan |  | ✅ |
| 2. Program berhasil dijalankan | ✅ |  |
| 3. Program dapat membaca masukan berkas .txt | ✅ |  |
| 4. Program dapat menghasilkan masukan secara acak |  | ✅ |
| 5. Solusi yang diberikan program optimal |  | ✅ |
| 6. Program dapat menyimpan solusi dalam berkas .txt |  | ✅ |
| 7. Program memiliki GUI | ✅ |  |

Refrensi

[Cyberpunk Hacking Explained and Breach Protocol Tips - Cyberpunk 2077 Guide - IGN](https://www.ign.com/wikis/cyberpunk-2077/Cyberpunk_Hacking_Explained_and_Breach_Protocol_Tips)

[Cyberpunk 2077 Breach Protocol guide - Polygon](https://www.polygon.com/cyberpunk-2077-guide-walkthrough/22163900/breach-protocol-encrypted-shard-militech-datashard-access-point-quickhack-buffer)

[Creating a brute-force algorithm in C++ - Stack Overflow](https://stackoverflow.com/questions/45423562/creating-a-brute-force-algorithm-in-c)

[performance - Brute-Force algorithm in C++ - Code Review Stack Exchange](https://codereview.stackexchange.com/questions/231696/brute-force-algorithm-in-c)

[Brute Force Algorithms Explained (freecodecamp.org)](https://www.freecodecamp.org/news/brute-force-algorithms-explained/)